Programación de robots inteligentes con autómatas y ROS

José María Cañas

josemaria.plaza@urjc.es

29 de septiembre de 2017

Introducción

- $robot = hardware + software$
- $hardware = sensores + actuadores + computadoras$

Software para robots

- La **inteligencia** de los robots radica en su **software**.
- $sw = middleware + aplicaciones$

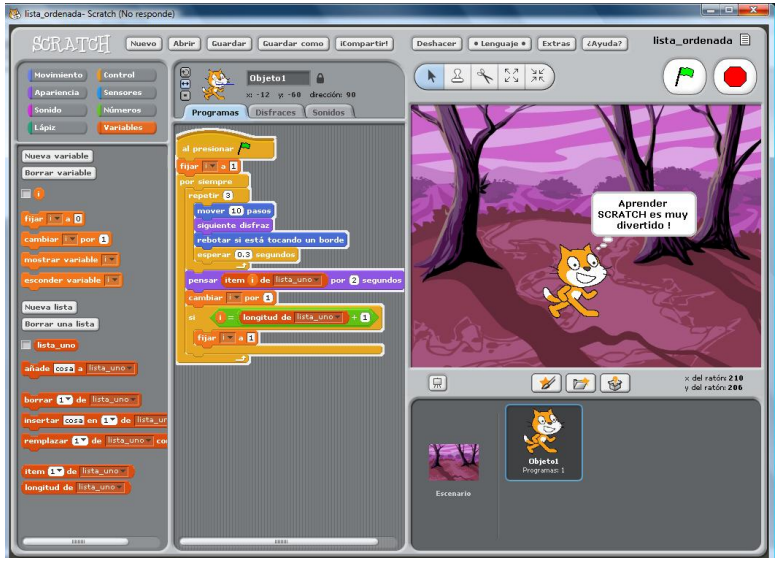
Middlewares

- $middleware = drivers + arquitecturaSW + herramientas + bibliotecas$
- arquitectura SW distribuida
- ROS, ICE, DDS...
- OpenCV, PCL, OMPL, nodos ya existentes...

Aplicaciones

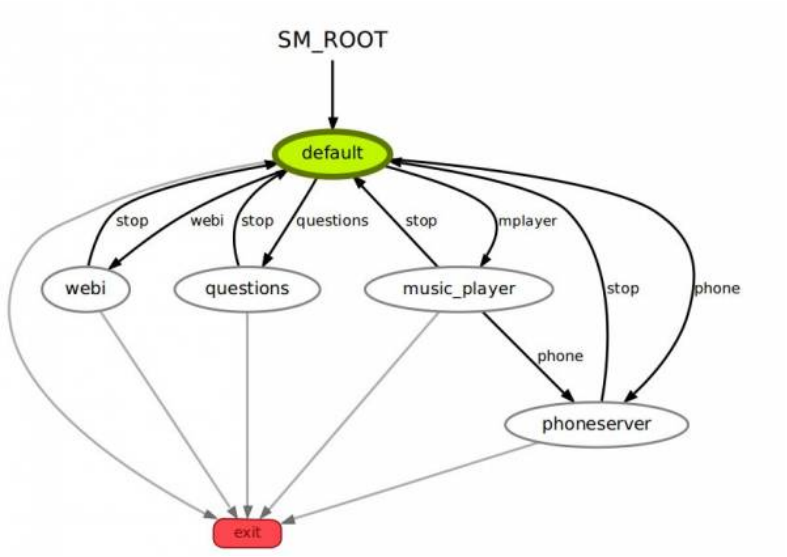
- $Aplicacion = percepcion + decisiones$
- Percepción: elaborar información, sensores
- Decisiones: control, planificación, actuación
- Algoritmos
- Lenguajes texto: C++, Python... y visuales

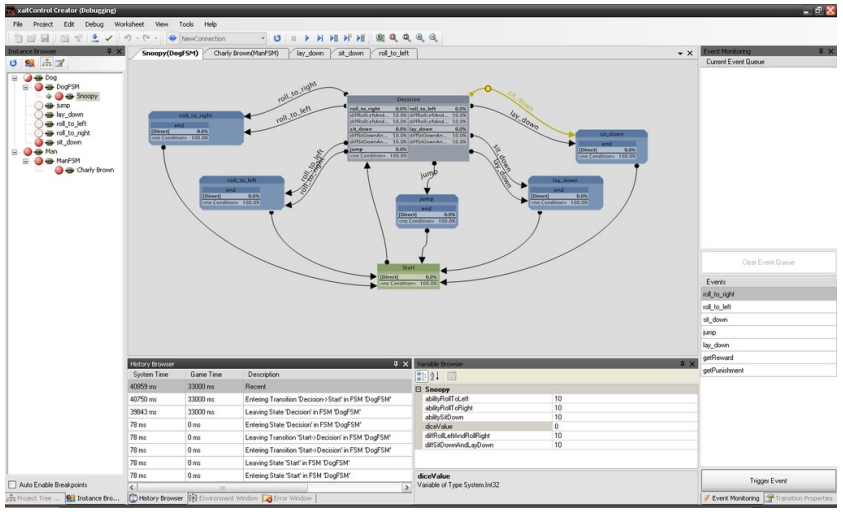
```
84  int segmentationType; //0 ninguna, 1 NITE
85  int mainFPS;
86
87
88  void* updateThread(void*)
89  {
90          //glfwInit();
91
92          libfreenect2::Freenect2 freenect2;
93          libfreenect2::Freenect2Device *dev = freenect2.openDefaultDevice();
94
95          if(dev == 0)
96          {
97                  std::cout << "no device connected or failure opening the default one
98                  return (NULL);
99          }
100         libfreenect2::SyncMultiFrameListener listener(libfreenect2::Frame::Color | 1:
101         libfreenect2::FrameMap frames;
102
103         dev->setColorFrameListener(&listener);
104         dev->setIrAndDepthFrameListener(&listener);
105         dev->start();
106
107         std::cout << "Device serial: " << dev->getSerialNumber() << std::endl;
108         std::cout << "Device firmware: " << dev->getFirmwareVersion() << std::endl;
109
110         float cycle=(float)(1/(float)mainFPS)*1000000;
```



Arquitectura, organización

- Sistemas reactivos
- Sistemas deliberativos
- Híbridos de tres capas
- Sistemas Basados en Comportamientos
- Autómatas de estado finito





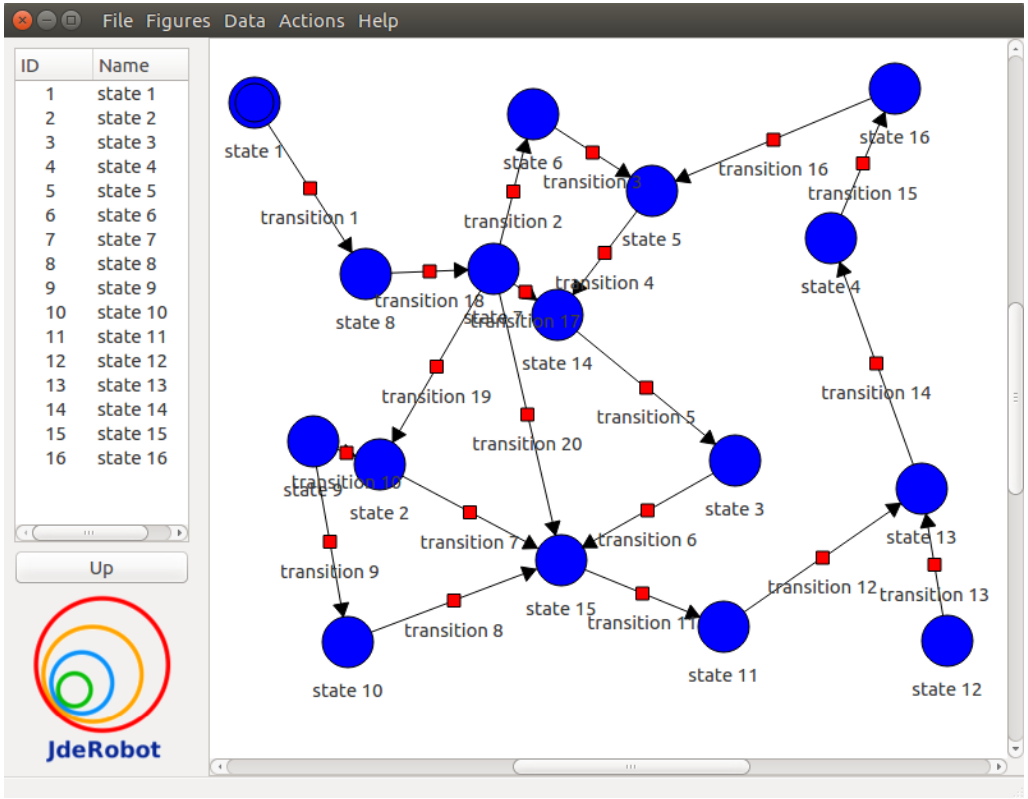
Herramienta VisualStates

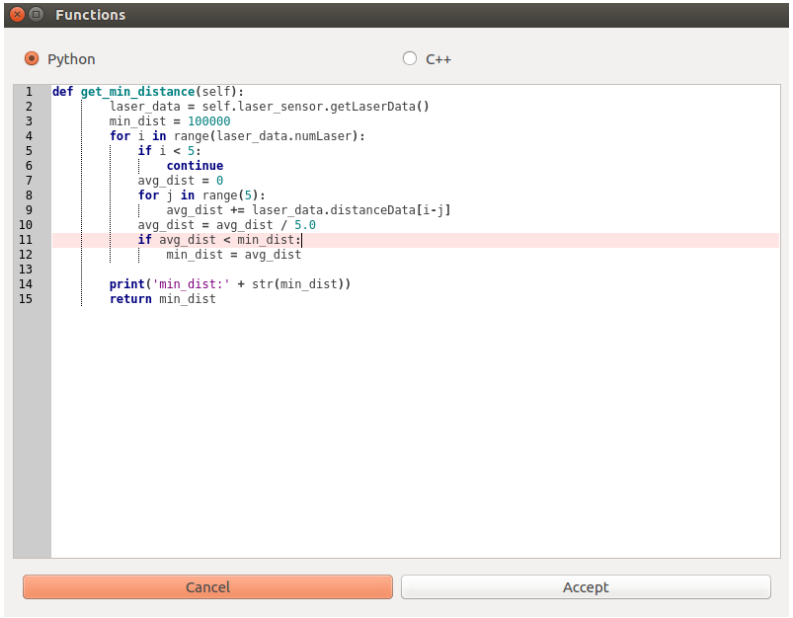
Diseño visual de la aplicación robótica con estados y transiciones

- Genera automáticamente nodo ROS (y ICE)
- Soporte para C++ y para Python
- GUI en ejecución, depuración
- Multinivel, escala en complejidad
- Software libre, dentro de JdeRobot
- Genera software más robusto y desarrollo más ágil

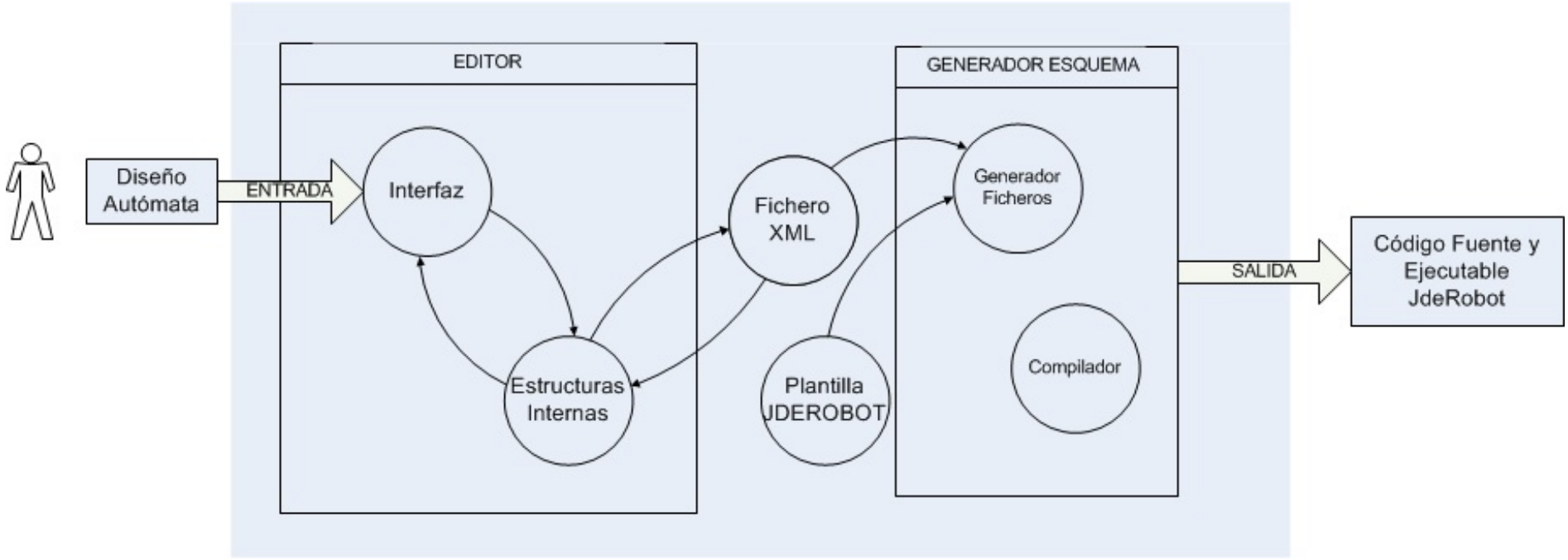
Editor

- Edición visual: estados y transiciones
- Ventanas de texto: código de estados, de transiciones y auxiliar



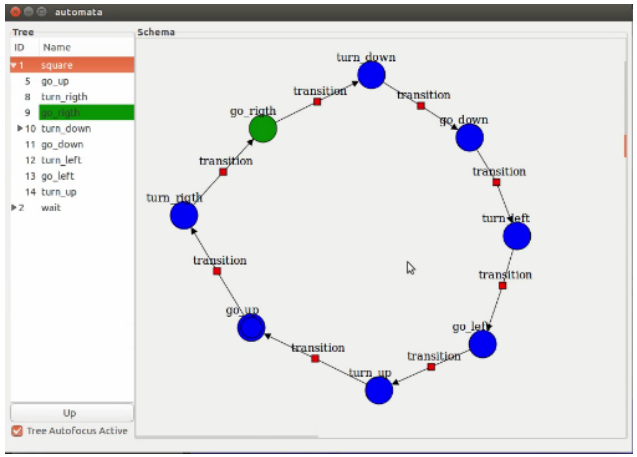


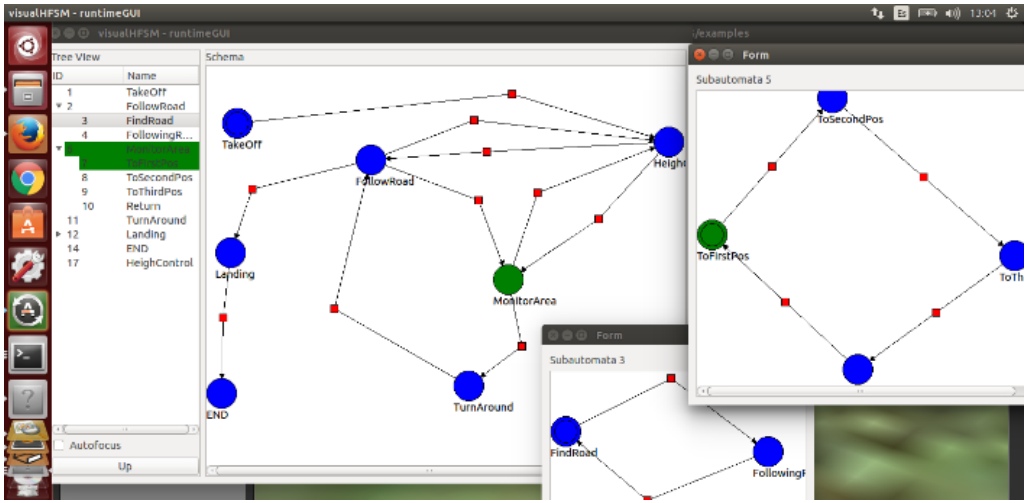
¿Cómo está hecho?



Plantillas

- Motor temporal iterativo
- Multinivel, jerarquía
- GUI en ejecución, depuración
- Conectable a sensores, actuadores y otros nodos





```
int Generate::init_py (){
    this->fs.open(this->path.c_str(), std::fstream::out);
    if (this->fs.is_open()){
        this->generateHeaders_py();
        this->generateAutomataClass_py();
        //CREATE GUI SUBAUTOMATAS
        this->generateMain_py();
        this->fs.close();

        this->fs.open(this->cfgpath.c_str(), std::fstream::out);
        if (this->fs.is_open()){
            this->generateCfg();
            this->fs.close();
        }

        std::string permission("chmod +x " + this->path);
        system(permission.c_str());
        return 0;
    }else{
        return -1;
    }
}

void Generate::generateAutomataClass_py(){
    this->fs << "class Automata():" << std::endl;
    this->fs << std::endl;
    this->generateAutomataInit_py();
    this->generateFunctions_py();
    this->generateStartThreads_py();
    this->generateCreateGuiSubautomataList_py();
    this->generateShutDown_py();
    this->generateRunGui_py();
    this->generateSubautomatas_py();
    this->generateConnectToProxys_py();
    this->generateDestroyIc_py();
    this->generateStart_py();
    this->generateJoin_py();
    this->generateReadArgs_py();
}
```

```
#!/usr/bin/python
#HEADERS
import Ice
from automatagui import AutomataGui, QtGui, GuiSubautomata
.....................
class Automata():
    def __init__(self): ...

    def startThreads(self): ...

    def createAutomata(self): ...

    def shutDown(self):

    def runGui(self): ...

    def subautomata1(self): ...

    def connectToProxys(self):

    def destroyIc(self): ...

    def start(self): ...

    def join(self): ...

    def readArgs(self): ...

if __name__ == '__main__':
    signal.signal(signal.SIGINT, signal.SIG_DFL)
    automata = Automata()
    try:
        automata.connectToProxys()
        automata.readArgs()
        automata.start()
        automata.join()

        sys.exit(0)
    except:
        traceback.print_exc()
        automata.destroyIc()
        sys.exit(-1)
```

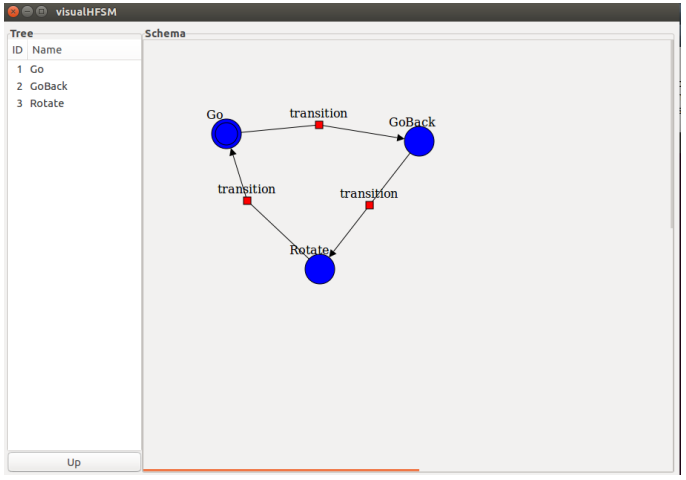
Últimas mejoras

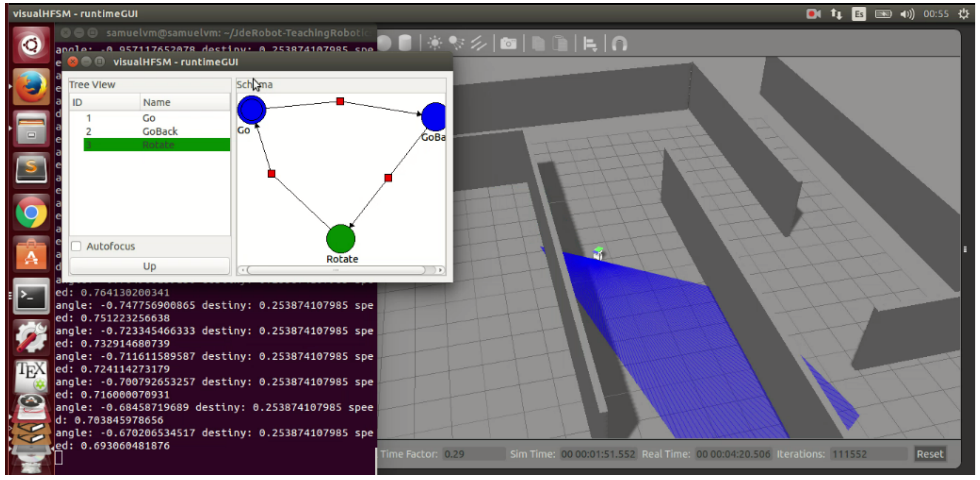
- Mejoradas plantillas con OOP
- Coloreado automático de texto
- Reescrita en Python
- Samuel Rey, Okan Asik (GSoC-2017)
- GitHub, integrada en paquete debian JdeRobot-5.6

Ejemplos

Choca-Gira

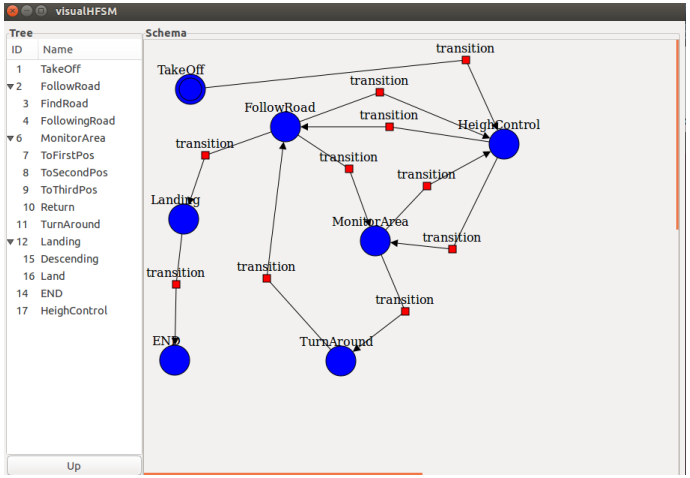
- Aplicación sencilla resuelta con un autómata mononivel.

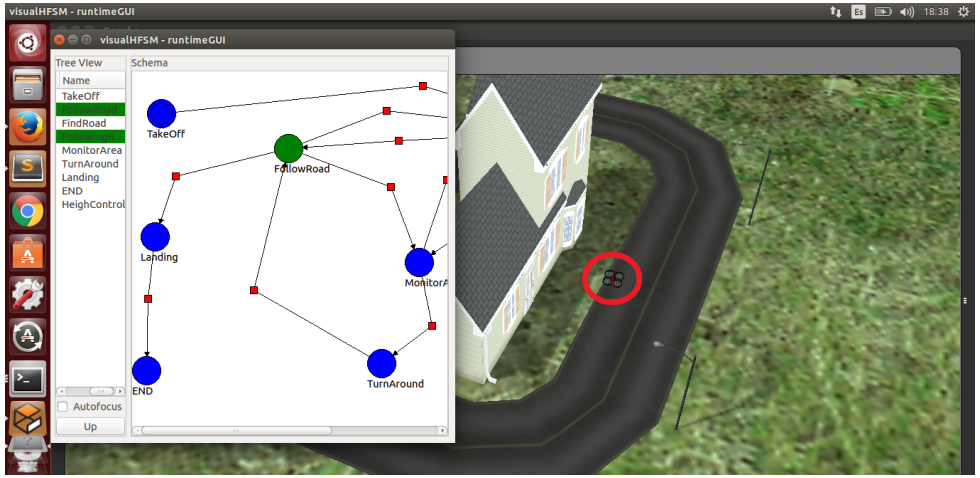




Monitorizar un área

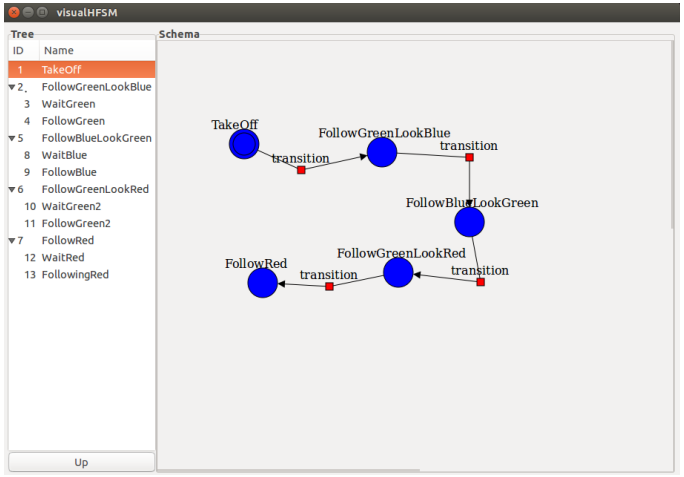
- Comportamiento más complejo.
- Autómata multinivel.

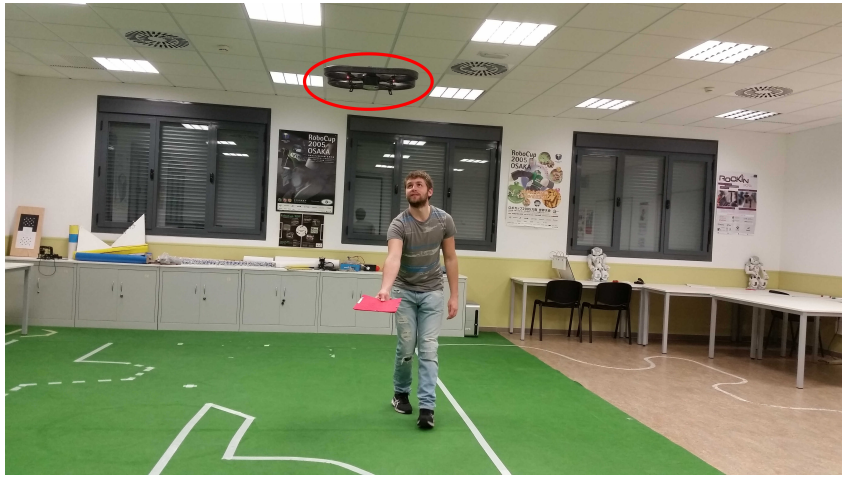




Sigue Colores

- Los componentes generados son compatibles con robots reales.
- Muestra la potencia del autómata frente a sistemas reactivos puros.





Conclusiones

- Herramienta para programar la inteligencia de un robot
- Potencia de los autómatas multinivel
- Editor gráfico de estados y transiciones
- Ventanas de texto
- Genera automáticamente nodo ROS